A

Project Report

On

PLC Implementation of Supervisory Control for a Dynamic Power Flow Controller using a Modular Approach

Submitted by

S.Sivaraman, Shishir Ramkrishna Joshi, Sudeep Maharana

(107EI006) **(107EI014)** **(107EI034)**

In partial fulfillment of the requirements for the degree in

Bachelor of Technology in Electronics & Instrumentation Engineering

Under the guidance of

Prof. Tarun Kumar Dan

Department of Electronics & Communication Engineering

National Institute of Technology Rourkela

May, 2011

CERTIFICATE

It is certified that the work contained in the thesis titled **"*Supervisory Control of Discrete Event System Model of a Dynamic Power Flow Controller using a Modular Approach*"** submitted by **Mr. S.Sivaraman, Mr. Shishir Ramkrishna Joshi** and **Mr. Sudeep Maharana**, has been carried out under my supervision and this work has not been submitted elsewhere for a degree.

_____

Date:

Mr. Tarun Kumkar Dan
Professor
Dept. of Electronics and Communication Engg.
NIT Rourkela

Acknowledgement

First and the foremost, we would like to offer our sincere gratitude to our thesis supervisor, **Prof. Tarun Kumar Dan** for his immense interest and enthusiasm on the project. His technical prowess and vast knowledge on diverse fields left quite an impression on us. He was always accessible and worked for hours with us and we always found his helping hand when it was required. He has been a constant source of encouragement for us. We also sincerely thank Prof. Max H. Queiroz, Federal University of Santa Catarina for his constant support and help by sharing his work with us. The software tool **ides2st** developed by his team has been of great importance to our work.

S.SIVARAMAN	SHISHI RAMKRISHNA JOSHI	SUDEEP MAHARANA
107EI006	107EI014	107EI034

ABSTRACT

Dynamic Power Flow Controller (DPFC) provides steady-state and dynamic power flow control for power lines and is considered as a Flexible AC Transmission System (FACTS) controller. This paper deals with control of a standard DPFC using a Discrete Event System model. The Supervisory Control of DES has been used to implement Modular supervisors for the DPFC. Despite the fact that the SCT is well consolidated, with a large number of publications focusing on the theoretical aspects, the industrial application is unknown. It is mainly due to the complexity of the theory. The numbers of states and events to be controlled are very large even for the seemingly simple systems. In recent years, a model for modular approach to the Supervisory Control for performing the formal synthesis of Supervisors has been proposed. Programmable Logic Controllers are used for the physical implementation of the controllers. Some problems in physical realization of Supervisors in PLCs are dealt with.

Keywords: Discrete Event Systems, Dynamic Power Flow controllers, Programmable Logic Controllers

List of Tables

Chapter 4

List of Figures

Chapter 3

Chapter 4

CONTENTS

CHAPTER 1

Introduction

1.1 Definition and Brief Review

Programmable logic controllers have been used extensively in industrial control applications since their advent in the 70s. The programming of logic controllers has been done majorly by the knowledge of the programmer and no formal methods are used. Hence, the task of writing the code becomes a difficult one with the efficiency of the code varying from programmer to programmer. The ladder logic structure of coding PLCs makes it difficult to realize higher level concepts such as function calls and looping. The discrete event based modeling of systems provides a suitable sequential structure to the programming of PLCs.

Many control problems in the industry, especially manufacturing processes, can be dealt as Discrete Event problems. The DES based modeling and Supervisory Control of a manufacturing cell is already dealt with in [1]. Hence, in our paper, we deal with another problem, that is, Supervisory Control of a Dynamic Power Flow Controller using the same approach as in [2]. The Supervisory Control Theory (SCT) was proposed first by Ramadge and Wonham [3]. In this paper, they introduce the concept of Supervisors and how a feedback system is established for the control of a DES. Although, SCT has gained critical acclaim in the academic sector, industrial applications have been minimal. This is mainly due to the fact that the number of states in the Supervisor increases exponentially and is so larger that the physical realization becomes impossible. A modular approach was suggested by Max and Vieira [4]. According to this

approach, instead of a single monolithic supervisor, a number of decentralized supervisors are used which work synchronously to achieve the same control action as the monolithic supervisor.

The physical implementations of the supervisors are somewhat complicated due the "bridge between the asynchronous DES world and synchronous PLC world" [5]. The DES modeling assumes the events to occur spontaneously at any random instant. But, the PLC follows a synchronous system of scan cycles. Some problems with physical implementation of supervisors are dealt with in [5], [6], [7]. In [5] , Fabian and Hellgren discuss some problems such as causality, inexact synchronization and avalanche effect. They discuss some possible solutions to t above problems which are to some extent implemented in our work. For physical implementation of the supervisors, they define a concept called interleave insensitivity and Malik [6] proposes an algorithm to verify if a supervisor satisfies this condition. Although all these problems have been discussed effective solutions are still open to research. Max and Vieira [4] suggest some methods while programming to overcome some of the problems discussed. They define a three level architecture which contains the Modular supervisors at the top, the product systems in the next level and the related operational procedures in the lowest level. They also use some auxiliary variables in the code to make sure the supervisors are not updated in the next cycles before the output is changed. Our work uses all these methods to implement the modular supervisors for a DPFC.

1.2 Research Objectives

As discussed earlier the SCT, although it has earned much acclaim in the academic sector, does not have much industrial applications as of yet. Hence, the major objective of our research is:

- to model a Discrete Event System, in our case a Dynamic Power Flow Controller.
- to develop supervisors to the system using SCT with a modular approach.
- to simulate the supervisors and the system to test proper functioning.
- automatic code generation for a PLC using ides2st.
- to address some implementation problems of SCT and suggest solutions.

Our work mainly aims at proving that the complete automation of the SCT based controller design process in possible in the future. By this paper, we hope to spread the advantages of SCT based modelling to conventional methods and finally aid the process of utilization of SCT in industrial control applications.

CHAPTER 2

Literature Review

Earlier Discrete-Event Systems were sufficiently simple that intuitive and ad-hoc solutions were sufficient. But, due to the increasing complexity of man-made systems, has taken such systems to such a level that formal methods for analysis and design are required. One of the first papers on the formal methods for control of discrete event systems were by Ramadge and Wonham [3]. The main advantage of the model is that it separates the concept of open loop dynamics (plant) from the feedback control, and thus permits the formulation and solution of a variety of control synthesis problems. After the initiative by them, many other researchers developed the theory to include concepts like controllability, observability, aggregation, and modular, decentralized, and hierarchical control. But, what the researches had not considered was the implementation of SCT.

In 1992, Balemi [8] proposed an interpretation of supervisory control theory from an input/output perspective. The plant was modeled as an input/output process accepting commands as inputs, and producing as outputs some messages regarding changes that occurred in the system. A controller controlling the system was described in a similar way, accepting the outputs of the plant, and in turn producing commands. Under these semantics both the controller and the plant formed the "generating" process in the closed-loop systems. This was in contrast to the original framework of Ramadge and Wonham where the plant alone was the "generator". Balemi also dealt with some problems of communication delay between the plant and the supervisor. Using this scheme a control environment for a Rapid Thermal Multiprocessor (RTM) had been

implemented at the Center for Integrated Systems at Stanford University. The environment provided both manual and automatic control.

Although the SCT attained critical acclaim in academia, its industrial applications were unknown. Fabian and Hellgren [5] in 1996, suggested that the main reason for this was the discrepancy between the abstract supervisor and its physical implementation. This is specifically noticeable when the implementation is supposed to be based on programmable logic controllers (PLCs), as is the case with many manufacturing systems. The asynchronous event-driven nature of the supervisor is not straightforwardly implemented in the synchronous signal-based PLC. They dealt with some problems in physical implementation of supervisors like simultaneity of events, inexact synchronization, causality and choice with examples. Malik [6] also dealt with problems like determinism and suggested some solutions.

In 2002 Max and Cury [1] attempted the modular supervisory control of a manufacturing cell. They suggest a three level structure for the PLC implementation of the supervisors. Again in 2006 [4], they improved their structure and proposed a Sequential Flow Chart based algorithm for developing the PLC code. In 2009 Max and Silva [2], developed the control scheme for a factory manufacturing cell using the methods in [4] and they also used an automatic code generator for the PLC code part. This was a real step forward in the automation of the controller implementation process. Further, the tools TCT and IDES developed by Wonham [9] and Rudie [10] respectively greatly aided to the ease of design process.

CHAPTER 3

Theory

3.1 Discrete Event Systems

A system is one of those primitive concepts whose understanding is best left to intuition. The IEEE Standard Dictionary of Electrical and Electronic Terms defines a system as a combination of components that act together to perform a function not possible with any of the individual parts. There are two salient points in this definition; firstly the system contains interacting components. Secondly, they perform specific functions. Systems can be classified into varies types on the basis of various criteria such as Static and Dynamic Systems, Time-varying and Time-invariant Systems, Linear and Nonlinear Systems, Continuous-State and Discrete-State Systems, Time-driven and Event-driven System, Deterministic and Stochastic Systems, Discrete-time and Continuous-time Systems.

When the state space of a system is naturally described by a discrete set like {0, 1, 2,...}, and state transitions are only observed at discrete points in time, we associate these state transitions with “events” and talk about a “discrete event system” [9]. An event should be thought of as occurring instantaneously causing a transition from one state to another. An event may be identified with a specific action taken (e.g., somebody presses a button). It may be viewed as a spontaneous occurrence dictated by nature (e.g., a computer goes down for whatever reason too complicated to figure out). Or it may be the result of several conditions which are suddenly all met.

Definition. A Discrete Event System (DES) is a discrete-state, event-driven system, that is, its state evolution depends entirely on the occurrence of asynchronous discrete events over time.

One of the formal ways to study the logical behavior of DES is based on the theories of languages and automata. The starting point is the fact that any DES has an underlying event set *E* associated with it. The set *E* is thought of as the "alphabet" of a language and event sequences are thought of as "words" in that language.

For real systems, E is finite. A sequence of events taken out of this alphabet forms a "word" or "string". A string consisting of no events is called the empty string and is denoted by ε. The length of a string is the number of events contained in it, counting multiple occurrences of the same event. If s is a string, its length is denoted by |s|. By convention, the length of the empty string ε is taken to be zero.

Language

A language defined over an event set E is a set of finite-length strings formed from events in E. As an example, let E = {a, b, g} be the set of events. A language may be defined as for this event set, L1= {ε, a, abb}

Operations on Languages:

The usual set operations, such as union, intersection, difference, and complement with respect to E∗, are applicable to languages since languages are sets. In addition, there are some more operations defined:

Concatenation:

Let L_a,$L_b \subseteq E*$,

then $L_aL_b := \{s \in E*:(s = s_as_b)$ and $(s_a \in L_a)$ and $(s_b \in L_b)\}$

In words, a string is in L_aL_b if it can be written as the concatenation of a string in L_a with a string in L_b.

Prefix-closure:

Let $L \subseteq E*$,

then $L := \{s \in E*:(\exists t \in E*)[s_t \in L]\}$

In words, the prefix closure of L is the language denoted by L and consisting of all the prefixes of all the strings in L. In general, $L \subseteq \overline{L}$ is said to be prefix-closed if $L = \overline{L}$. Thus language L is prefix-closed if any prefix of any string in L is also an element of $\overline{L}$.

Formal Definition of a DES

A discrete Event System maybe define by a Deterministic automaton denoted by G, is a six-tuple

$G = (X, E, f, \Gamma, x_0, X_m)$

where:

X is the set of states

E is the finite set of events associated with G

$f : X \times E \rightarrow X$ is the transition function: f(x, e)= y means that there is a transition labeled by event e from state x to state y; in general, f is a partial function on its domain.

$\Gamma: X \rightarrow 2^E$ is the active event function (or feasible event function); Γ(x)is the set of all events e for which f(x, e) is defined and it is called the active event set (or feasible event set) of G at x.

x_0 is the initial state.

$X_m \subseteq X$ is the set of marked states.

The automaton G operates as follows. It starts in the initial state x_0 and upon the occurrence of an event $e \in \Gamma(x_0) \subseteq E$ it will make a transition to state $f(x_0,e) \in X$. This process then continues based on the transitions for which **f** is defined.

Languages generated and marked

The language generated by $G =(X,E, f, \Gamma,x_0,X_m)$is

$$\boldsymbol{L(G) := \{s \in E* : f(x0, s) \text{ is defined}\}}$$

The language marked by G is

$$\boldsymbol{Lm(G) := \{s \in L(G): f(x0, s) \in Xm\}}$$

Blocking

Automaton G is said to be blocking if

$$\boldsymbol{Lm(G) \subset L(G)}$$

where the set inclusion is proper, and non-blocking when

$$\boldsymbol{Lm(G) = L(G)}$$

Unary Operations

Now let's consider operations that alter the state transition diagram of an automaton. The event set E remains unchanged.

Accessible Part

From the definitions of L(G)and $L_m(G)$, we can delete from G all the states that are not accessible or reachable from x_0 by some string in L(G), without affecting the languages generated and marked by G. When a state is "deleted", all the transitions that are attached to that state are also deleted.

$Ac(G):=(X_{ac},E,f_{ac},x_0,X_{ac,m})$ where

Ac stands for accessible part of G.

$X_{ac} = \{x \in X :(\exists s \in E^*)[f(x_0,s)= x]\}$

$X_{ac,m} = X_m \cap X_{ac}$

$f_{ac} = f|X_{ac}\times E\rightarrow X_{ac}$

The notation $f|X_{ac}\times E\rightarrow X_{a}c$ means that 'f' is restricted to the smaller domain of the accessible states X_{ac}.

Clearly, the Ac operation has no effect on L(G)and $L_m(G)$. Thus, without loss of generality, an automaton is accessible, that is, G = Ac(G).

Co-accessible Part

A state x of G is said to be co-accessible to X_m, or simply co-accessible, if there is a path in the state transition diagram of G from state x to a marked state. The operation of deleting all the states of G that are not coaccessible is denoted by CoAc(G), where CoAc stands for taking the "coaccessible" part.

$CoAc(G):=(X_{coac},E,f_{coac},x_{0,coac},X_m)$

Where

$X_{coac} = \{x \in X :(\exists s \in E^*)[f(x, s) \in X_m]\}$

$x_{0,coac} = x_0$ if $x_0 \in X_{coac}$

undefined otherwise

$f_{coac} = f|X_{coac}\times E\rightarrow X_{coac}$

The CoAc operation may shrink L(G), since it may involve deleting states that are accessible from x_0; however, the CoAc operation does not affect $L_m(G)$, since a deleted state cannot be on

any path from x_0 to X_m. If G = CoAc(G), then G is said to be coaccessible; in this case, L(G)= L_m(G).

Coaccessibility is closely related to the concept of blocking; since an automaton is said to be blocking if L(G) = L_m(G). Therefore, blocking necessarily means that L_m(G) is a proper subset of L(G) and consequently there are accessible states that are not coaccessible.
If the CoAc operation results in $X_{coac} = \emptyset$ (this would happen if $X_m = \emptyset$ for instance), an empty automaton is obtained.

Trim Operation

An automaton that is both accessible and coaccessible is said to be trim. Trim operation is defined to be

Trim(G):= CoAc[Ac(G)] = Ac[CoAc(G)]

where the commutativity of Ac and CoAc is easily verified.

Projection and Inverse Projection

Let G have event set E. Consider $E_s \subset E$. The projections of L(G)and L_m(G)from E^* to E^*_s , P_s[L(G)] and P_s[L_m(G)], can be implemented on G by replacing all transition labels in $E \setminus E_s$ by ε. The result is a nondeterministic automaton that generates and marks the desired language.

Composition Operations

Two operations on automata are defined: product, denoted by ×, and parallel composition, denoted by ||. Parallel composition is often called synchronous composition and product is

sometimes called completely synchronous composition. These operations model two forms of joint behavior of a set of automata that operate concurrently.

Product

The product of G1 and G2 is the automaton

$G1 \times G2 := Ac(X_1 \times X_2, E_1 \cup E_2, f, \Gamma 1 \times 2, (x_{01}, x_{02}), X_{m1} \times X_{m2})$

where

$$f((x1, x2), e) := \begin{cases} (f1(x1, e), f2(x2, e)) \text{ if } e \in \Gamma 1(x1) \cap \Gamma 2(x2) \\ \text{undefined} \qquad \text{otherwise} \end{cases}$$

and thus $\Gamma_{1\times2}(x_1,x_2)=\Gamma_1(x_1) \cap \Gamma_2(x_2)$.

Properties of product

1. Product is commutative up to a reordering of the state components in composed states.

2. Product is associative i.e. $G_1 \times G_2 \times G_3 = (G_1 \times G_2) \times G_3 = G_1 \times (G_2 \times G_3)$

Parallel Composition

The parallel composition of G1 and G2 is the automaton

$G1 \parallel G2 = Ac(X_1 \times X_2, E_1 \cup E_2, f, \Gamma_{1\parallel2}, (x_{01}, x_{02}), X_{m1} \times X_{m2})$

where

$$f((x1, x2), e) = \begin{cases} (f1(x1, e), f2(x2, e)) \text{ if } e \in \Gamma 1(x1) \cap \Gamma 2(x2) \\ (f1(x1, e), x2) \text{if } e \in \Gamma 1(x1) \setminus E2 \\ (x1, f2(x2, e)) \text{if } e \in \Gamma 2(x2)\ E1 \\ \text{undefined otherwise} \end{cases}$$

and thus $\Gamma_{1\parallel2}(x_1, x_2)=[\Gamma_1(x_1) \cap \Gamma_2(x_2)] \cup [\Gamma_1(x_1) \setminus E] \cup [\Gamma_2(x_2) \setminus E]$.

Properties of parallel composition:

1. The set inclusion

$P_i[L(G_1||G_2)] \subseteq L(G_i)$, for i =1, 2

The same result holds for the marked language. The coupling of the two automata by common events may prevent some of the strings in their individual generated languages to occur, due to the constraints imposed in the definition of parallel composition regarding these common events.

2. Parallel composition is commutative up to a reordering of the state components in composed states.

3. It is associative:

$$(G1||G2)||G3 = G1||(G2||G3)$$

3.2 Supervisory Control Theory

Consider a DES modeled by a pair of languages, L and L_m , where L is the set of all strings that the DES can generate and $L_m \subseteq L$ is the language of marked strings that is used to represent the completion of some operations or tasks; the definition of Lm is a modeling issue. L and L_m are defined over the event set E. L is always prefix-closed, that is, $L = \overline{L}$, while L_m need not be prefix-closed. Assume that L and L_m are the languages generated and marked by automaton

$G = (X, E, f, \Gamma, x_0 , X_m)$

where X need not be finite; that is,

$L(G) = \overline{L}$ $\quad$ $L_m (G) = L_m$

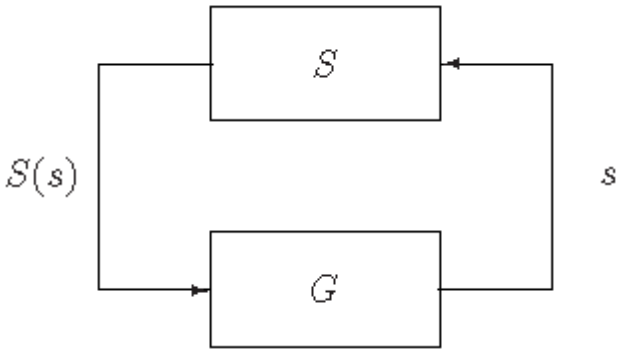


Fig 3.2.1

A supervisor S is to be attached to interact with G in a feedback manner as depicted in the Fig 3.2.1.

The event set E can be divided into two sets

$$E = Ec \cup E_{uc}$$

Where

E_c is the set of controllable events: these are the events that can be disabled by supervisor S.

E_{uc} is the set of uncontrollable events: these events cannot be disabled by supervisor S.

There are many reasons why an event would be modeled as uncontrollable: it is inherently unpreventable, it cannot be prevented due to hardware or actuation limitations, or it is modeled as uncontrollable by choice, as for example when the event has high priority and thus should not be disabled or when the event represents the tick of a clock.

Assuming that all the events in E executed by G are observed by supervisor S, in Fig. 3.2.1, s is the string of all events executed so far by G and s is entirely seen by S. The control paradigm is as follows. The transition function of G can be controlled by S in the sense that the controllable events of G can be dynamically enabled or disabled by S.

Formally, a supervisor S is a function from the language generated by G to the power set of E:

$$S : L(G) \rightarrow 2E$$

For each s ∈ L(G) generated so far by G (under the control of S),

$$S(s) \cap \Gamma(f(x_0, s))$$

is the set of enabled events that G can execute at its current state f (x_0 , s). In other words, G cannot execute an event that is in its current active event set, Γ(f (x_0 , s)), if that event is not also contained in S(s). In view of the partition of E into controllable and uncontrollable events, we could say that supervisor S is admissible if for all s ∈ L(G)

$$E_{uc} \cap \Gamma(f(x_0, s)) \subseteq S(s)$$

which means that S is not allowed to ever disable a feasible uncontrollable event.

Languages generated and marked by S/G

The language generated by S/G is defined recursively as follows:

1. ε ∈ L(S/G)
2. [(s ∈ L(S/G)) and (s_σ ∈ L(G)) and (σ ∈ S(s))] ⇔ [s_σ ∈ L(S/G)] .

The language marked by S/G is defined as follows:

$$L_m(S/G) := L(S/G) \cap L_m(G)$$

Clearly, L(S/G) ⊆ L(G) and it is prefix-closed by definition. As for L_m (S/G), it consists exactly of the marked strings of G that survive under the control of S. Overall, we have the set inclusions

$$\emptyset \subseteq L_m(S/G) \subseteq L_m(S/G) \subseteq L(S/G) \subseteq L(G)$$

The empty string ε is always in L(S/G) since it is always contained in L(G); here, we have excluded the degenerate case where G is the empty automaton. Thus, when we adjoin S to

G, G is to start in its initial state at which time its possible first transition will be constrained by the control action S(ε).

To Prevent a supervisor from being blocking(live block/ dead block), we define a blocking and non-blocking supervisor as follows-

Blocking in controlled system

The DES S/G is blocking if

$$L(S/G) = L_m (S/G)$$

and non-blocking when

$$L(S/G) = L_m (S/G)$$

Since the blocking properties of S/G are as much the result of S as of the structure of G, supervisor S controlling DES G is blocking if S/G is blocking, and supervisor S is non-blocking if S/G is non-blocking.

Modeling of Specifications as Automata

In most of the cases, the construction of the automaton that generates/marks the applicable language requirement, say L_a, is preceded by the construction of a simple automaton that captures the essence of the natural language specification. Let this automaton be H_{spec} . We then combine H_{spec} with G, using either product or parallel composition, as appropriate, to obtain H_a where

$$L(H_a) = L_a$$

In the case of several natural language specifications, there will be many $H_{spec,i}$, i = 1, . . . , n, that will be combined with G.

The choice of product or parallel composition to compose H_{spec} with G is based on the events that appear in the transition diagram of H_{spec} and on how we wish to define the event set of H_{spec} :

- If the events that can be executed in G but do not appear in the transition diagram of H_{spec} are irrelevant to the specification that H_{spec} implements, then we use parallel composition and define the event set of H_{spec} to be those events that appear in its transition diagram.
- On the other hand, if the events that can be executed in G but do not appear in the transition diagram of H_{spec} are absent from H_{spec} because they should not happen in the admissible behavior L_a, then product composition operation is used. Equivalently, parallel composition can still be used provided the event set of H_{spec} is defined carefully and includes the events that should be permanently disabled.

In most cases, all the states of H_{spec} will be marked so that marking in H_a will be solely determined by G.

Illegal States

If a specification identifies certain states of G as illegal, then it suffices to delete these states from G, that is, remove these states and all the transitions attached to them, and then do the Ac operation to obtain Ha such that $L(H_a) = L_a$. If the specification also requires non-blocking behavior after the illegal states have been removed, then we do the Trim operation instead of the Ac operation and obtain Ha such that $L_m(H_a) = L_{am}$ and $L(H_a) = L_{am}$

Controllability Theorem

Consider DES G = (X, E, f, Γ, x_0) where $E_{uc} \subseteq E$ is the set of uncontrollable events. Let $K \subseteq L(G)$, where $K = \emptyset$. Then there exists supervisor S such that L(S/G) = K if and only if

$$KE_{uc} \cap L(G) \subseteq K$$

This condition on K is called the controllability condition.

Controllability

Let K and M = $\overline{M}$ be languages over event set E. Let E_{uc} be a designated subset of E. K is said to be controllable with respect to M and E_{uc} if

$$KE_{uc} \cap M \subseteq K$$

By definition, controllability is a property of the prefix-closure of a language. Thus K is controllable if and only if K is controllable. The language expression for the controllability condition can be rewritten as follows:

for all $s \in K$, for all $e \in E_{uc}$, $s_e \in M \Rightarrow s_e \in K$

3.3 Dynamic Power Flow Controller

A Dynamic Power Flow Controller is used in power flow control in transmission lines and is considered a Flexible AC Transmission (FACTS) device. A DPFC consists of a PST (Phase Shifting Transformer) and series of TSCs (Thyristor Switched Capacitor). We will not discuss the working principle of the PST and TSC in detail, but it is important to note that due to their inherent large time constants, PST does not provide very good dynamic control. Hence,

TSC which have very quick response are used for dynamic control and as PST provide a better steady-state response, they are used for static control. Hence, our control modeling should first take care of switching the TSC and then the PST. We will discuss the features of a standard DPFC for flow control as in [10]. We wish to model the same DPFC specifications as in [10]. The DPFC consists of:

- a conventional 115-MW (mechanically-switched) Phase-Shifting Transformer (PST) that can inject a quadrature voltage up to Vp=0.26pu, with 19 steps of 2-kV, and consequently introduces a maximum of 15° phase-shift, which can inject a lead/lag quadrature-phase voltage.
- a three-module series-connected multi-module Thyristor-Switched Capacitor (TSC) with reactance of XC1=4Ω, XC2=8Ω and XC3=12Ω, which provides seven steps corresponding to XCeq = 0, 4, 8, 12, 16, 20 and 24Ω. TSC can insert series capacitive reactance in 7 discrete steps to adjust the line series reactance.
- a 2×25-MVAr shunt-connected Mechanically-Switched Capacitor (MSC) at bus j to supply the required reactive power.

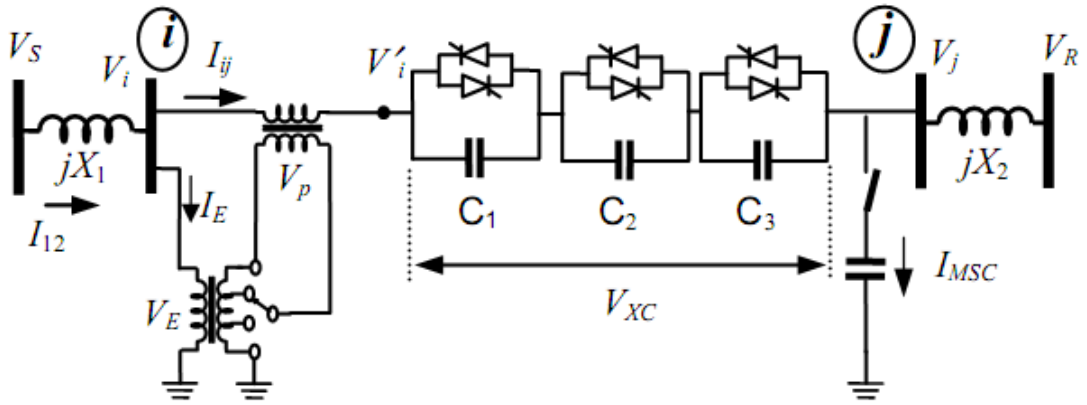


Fig.3.3.1 One-Line diagram of system

The TSC consists of three capacitors, which using the thyristors, give 7 combinations. We can regulate the reactive power by switching between these combinations. When an increase in power supply is demanded, the effective capacitance must be increased so that the reactive power increases. When power supply has to be decreased, a combination is chosen such that the effective capacitance is lowered.

The PST regulates the power supply in 19 steps. These tap-steps must be increased for an increase in power supply and decreased for decreasing power supply. When power supply has to be either lowered or increased, the TSC is switched to meet the requirements. As the PST tap-steps are increased/decreased, corresponding action is taken in the TSC so that the power supply is maintained at a constant level. In the ideal case, the power supply is at such a value that the TSC is in the default position. This is to ensure that a sudden change in demand is met with quickly.

The TSC and PST regulate power supply on the event of occurrence of certain signals from the powermeter. Moreover, the switching of the TSC and PST between different levels has to be controlled. Hence a Discrete Event System model is apt to describe the system accurately.

3.4 Programmable Logic Controllers

A programmable logic controller, commonly known as PLC, is a solid state, digital, industrial computer. It was invented in order to replace the sequential relay circuits which are mainly used for machine control. Applications of PLCs in automation are in the field of sequence control, motion control, process control, data management, and communication. Majority of the PLC applications are still utilized in machine control, material handling, sequence control applications; but on the other hand the number of process control applications that make use of

PLC are also in the rise. Apart from performing basic functions like sequential operation, timing and counting, PLCs can perform arithmetic operations as well. PLCs also have special modules for analog control functions like P, PI PID positioning control. PLCs can provide information on alarm limit detection, alarm messages, machine malfunction, production summary, machine status, etc. Intelligent modules with onboard microprocessors are also available.

Operation of a PLC

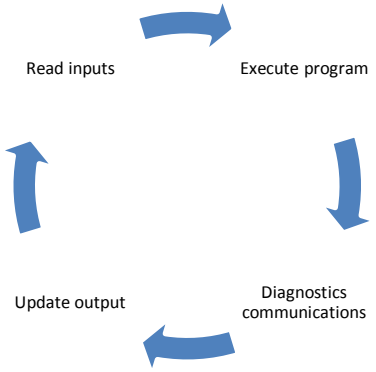


Fig. 3.4.1

Figure3.4.1 shows the steps in PLC operation cycle. During program execution, the processor reads all the inputs, and according to control application program energizes or de-energizes the outputs. Once all the logic has been solved, the processor will update all the outputs. The process of reading the inputs, executing the control application program, and updating the output is known as *scan*. During a scan operation, the processor also performs housekeeping tasks.

The inputs to the PLC are sampled by processor and the contents are stored in memory. Control program is executed. The input values stored in memory are used in control logic calculations to determine the values of outputs. The outputs are then updated. The cycle consisting of reading of inputs, executing the control program, and actuating the outputs is

referred to as ‘scan’, and the time taken to perform scan is called ‘scan time’. The speed at which PLC scans the memory depends on clock speed of CPU. The time to scan a program depends on the following parameters-

- Scan rate
- Length of the program
- Types of functions used in the program
- Faster scan time implies the inputs and outputs are updated frequently. Due to advanced technique of ASIC within the microcomputer for specific functions, scan times of different PLCs have reduced greatly.

Relay Logic and Ladder Logic

A relay is a simple device that uses a magnetic field to control a switch. When a voltage is applied to the input coil, the resulting current creates magnetic field. The magnetic field pulls a metal switch towards it and the contacts touch, closing the switch. The contact that closes when the coil is energized is called normally open (NO). The normally closed (NC) contacts close when the input coil is not energized and open when the input coil is energized.

The Ladder logic in the PLC is actually a computer Program that the user can enter and change. The ladder diagram language is basically a symbolic set of instructions used to create the controller program. These symbols are arranged to obtain the desired control logic that is to be entered into the memory of the PLC. A ladder diagram consists of individual rungs just like a real ladder. A line showing an input or several inputs and an output is known as a rung. Ladder logic programing is a graphical representation of the program designed to look like relay logic. The many similarities between the ladder diagrams used to program PLCs and the relay ladder logic formerly used to control industrial systems eased the transition from hardwired relay

systems to PLC-based systems. The ability to monitor PLC logic in ladder diagram format also made troubleshooting easier for those already familiar with relay -based control systems.

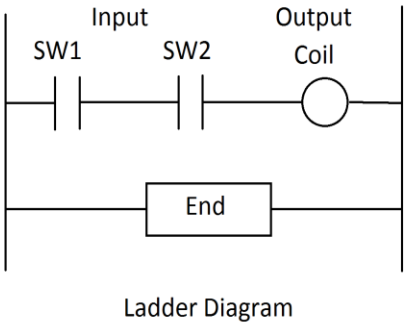


Fig 3.4.2

3.5 Problems in PLC Implementation of Supervisors

At first it may seem easy to implement the supervisors in a PLC. It looks like merely a matter of making the PLC behave as a state machine. But, there are a number of problems in implementing the supervisors in sequential, synchronous device like a PLC. The straight forward way to represent a Supervisor is to assign variables to each state and event and represent the event transitions with a logical AND between the state and the even variables. After the transition, the next state is set and the previous state is reset. Problems like initialization and many-to-one transitions are easily solved, but there are a number of more intricate problems.

1. Avalanche Effect

Signals always exist with different values, but events only exist momentarily. The events could be associated with rising edges of the signals. The rising edge can be detected in two scan cycles, i.e. if the event was low in the previous cycle and high in the current one, a rising edge

has occurred. But, associating events with rising and falling edges may lead to the avalanche effect.

The avalanche effect makes the program skip an arbitrary number of states during the same scan cycle. This is a result of the sequential execution of the program. This effect can be undone in some cases as in [5] using particular case by reversing the order of the code. But, in more complex FSMs, it might not be possible to perceive the order correctly. Also, if there are more than one states skipped, this solution might not work. We have used some techniques to resolve this like using auxiliary variables which will be discussed in detail in Chapter 5.

2. Simultaneity of events

Another problem while moving from event-based world to signal based world is simultaneity. Events are assumed to not to occur simultaneously, but signals on the other hand can very well occur simultaneously. We can guarantee the non-overlapping of events if we could detect the rising edges when they occurred. But, this is not possible due to the cyclic execution of the PLC. If the rising edges of the two events occur within the same scan cycle of the PLC, the two simultaneous events are detected. This problem cannot be remedied by programming, since it is the problem with the synchronous nature of the PLC, whereas the state machine implies an asynchronous execution. To remedy this, the supervisor should be such that it should depend on the different interleavings of the same events.

Interleave Insensitivity

A supervisor S is interleave insensitive with respect to a plant P and a sub-alphabet $\Sigma' \in \Sigma_p$ if for $s_1, s_2 \in (\Sigma_p - \Sigma')^*$ and $\sigma' \in \Sigma'$.

$ss_1s_2\sigma' \in L(P||S) \Rightarrow s(s_1|||s_2)\sigma' \subseteq L(P||S)$

This definition means that after any interleavings of the strings s_1 and s_2, the control decision is same. Typically, Σ' represents the events generated by the supervisor and $(\Sigma_p\text{-}\Sigma')^*$ represents the events generated by the plant. Since all interleavings lead to the same decision, the supervisors need only one of them and hence supervisor reduction is possible.

But, in general in real systems supervisors are not interleave insensitive. The order of the events is important. Hence the problem of simultaneity will exist in general.

3. Choice

The supervisor generated by the SCT is generally required to be minimally restrictive; that is, it allows the plant the greatest possible freedom while still upholding the specification. However, when the controller generates some events only one of the possible events must be generated. Generating several may be contradictory and catastrophic. In that case, the supervisor and the plant are entirely out of synch and there is no longer any guarantee that the supervisor, as implemented, can control the system satisfactorily. To resolve this problem the implementation must simultaneously choose and transit; and only a single event must be chosen. If the choice is not explicitly made by the implementor, the PLC itself will make the choice, determined by the ordering of the rungs.

4. Inexact synchronization

When a PLC executes the program, the plant is not observed, i.e. inputs are not being read. The scan cycle is assumed to be short in comparison to the plant response time. But, this is not always the case. A signal may occur in the plant while the execution of the program. The supervisor can have knowledge of commands and events that have occurred so far. Hence, if an signal occurs during the scan cycle it may lead to synchronization problems. Balemi [8] introduced the concept of delay insensitive languages.

A language K is said to be delay insensitive if, for $s \in K$, $\sigma_c \in \Sigma_s$, $\sigma_u \in \Sigma_p$, $s\sigma_p, s\sigma_u \in \overline{K} \Rightarrow s\sigma_u\sigma_c, s\sigma_c\sigma_u \in \overline{K}$.

Σ_s denotes the events generated by the supervisor while Σp are the events generated by the plant.

This definition captures delays in one direction only; from the plant to the supervisor. We can always control the plant so that only a single command is generated between the responses. The supervisor generates the command, and then waits for the responses. With a PLC implementation, this is natural, since the delay is due to the cyclic execution of the PLC. There are typically no delays in the direction from PLC to plant, since all communication is achieved through digital I/O.

Also, the definition concerns only delays of length one. This is assumed by Balemi [8] in order to simplify the problem. The delay is due to something happening in the plant while the PLC is executing its program. The next scan cycle, though, starts with an update of the signals, and thus, the event is seen, assuming that the scan cycle time is short compared with the time constants of the plant. Hence, the assumption is generally valid.

We had briefly discussed dome problems in PLC execution of supervisors. All these problems are discussed in detail with examples in [5] . Readers should go through [5] if they want some examples to help better understanding of the concepts.

CHAPTER 4

Modeling

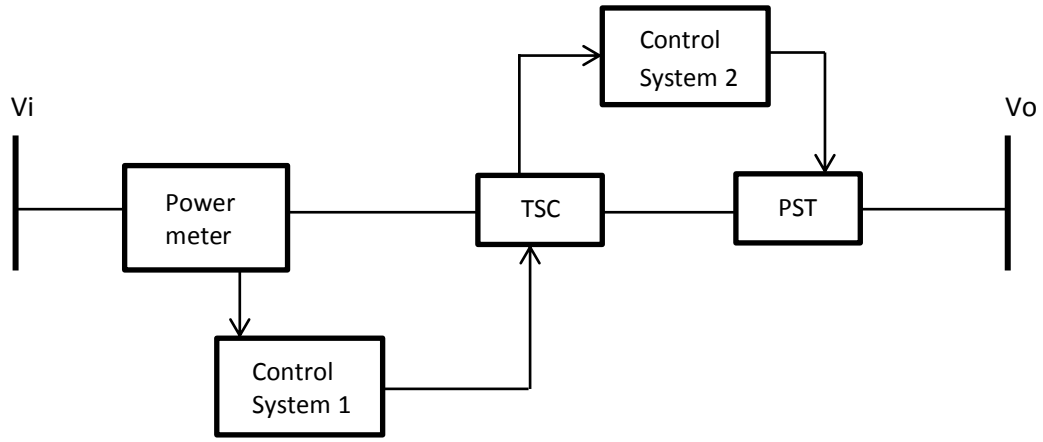


Fig.4.1. Block doagram of the modular supervisors

The DPFC can be thought of as being the composition of individual Power meter, TSC and PST plants. In addition, there can be two control systems, Control System 1and Control System 2. These blocks control the interaction between Power-meter & TSC and TSC and PST respectively. It is this interpretation of the DPFC that is used for modeling the supervisors. The modeling for the DPFC is done using generators in accordance with Ramadge and Wonham [3] .The automata for these generators are represented graphically by state transition diagrams. In these diagrams, all vertices (circles) represent a state, and concentric circles indicate a marked state. The transition labels indicate the event that causes the transition. The generators for the plants were designed in the graphical form. The modeling tool TCT [11] was used as an aid. The physical behavior of the Power-meter, TSC and PST plants were modeled separately as

PWRMTR, TSC_P & PST_P and the interaction between the plants were not taken into account for this step. The desired behavior of the system, taking into account their interactions, was also modeled as TSC_SPEC, PST_SPEC, CONTROL_SYSTEM_1_SPEC and CONSTROL_SYSTEM_SPEC. These automatons were developed using TCT. The following events were used as triggers for initiating transitions in the discrete event systems. Events labeled with even numbers are uncontrollable events, and events labelled with odd numbers are controllable events-

1. **Powermeter**

Event	**Event label**
Power meter initialized	11
Report decrease in power demanded	10
Report normal situation	12
Report increase in power demanded	14

2. **TSC**

Event	**Event label**
Capacitor decrease command	31
Capacitor decrease successful	32
Capacitor increase command	33
Capacitor increase successful	34

Capacitor decrease/increase failed	30

3. PST

Event	**Event Label**
Tap down command	41
Tap down successful	42
Tap up command	43
Tap up successful	44
Tap up/down failed	40

The control requirements being that if there was a increase in power demanded by powermeter (evnt_14), then the TSC is switched first i.e. evt_33 is enabled by the supervisor (but actually the supervisor disables all other controllable events other than evt_33). Then, if a normal situation is reported, the TSC is brought back and the PST is tapped up to maintain power at that level. And if a decrease in power is demanded the TSC is switched down first then, in the next cycle, the PST is tapped down and the TSC is brought back to position. If a failure event (evt_30, evt_40) occurs then Manual mode is switched on. The TSC is always brought back to state 0 for better range of dynamic control.

The state diagrams for these plants were generated using TCT.

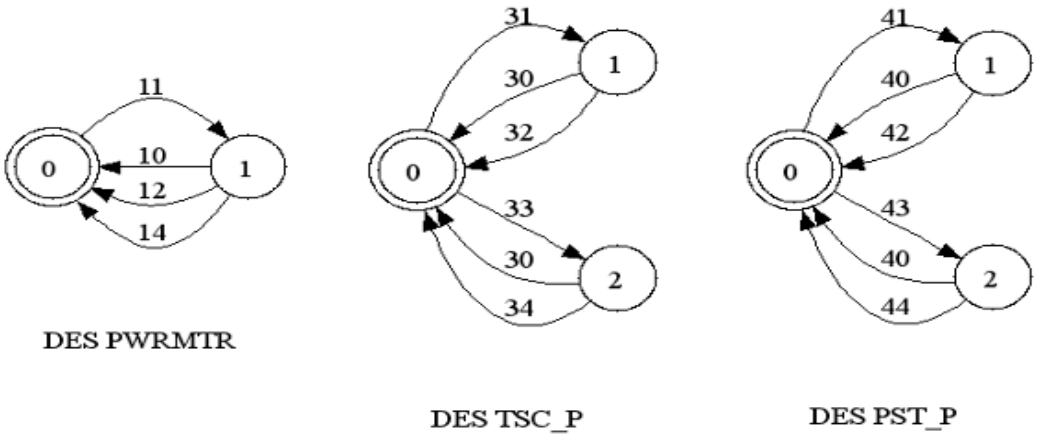


Fig.4.2. Generators of PLANTS G_{PWRMTR}, powermeter (left), G_{TSC},TSC (middle) and G_{PST}, PST (right)

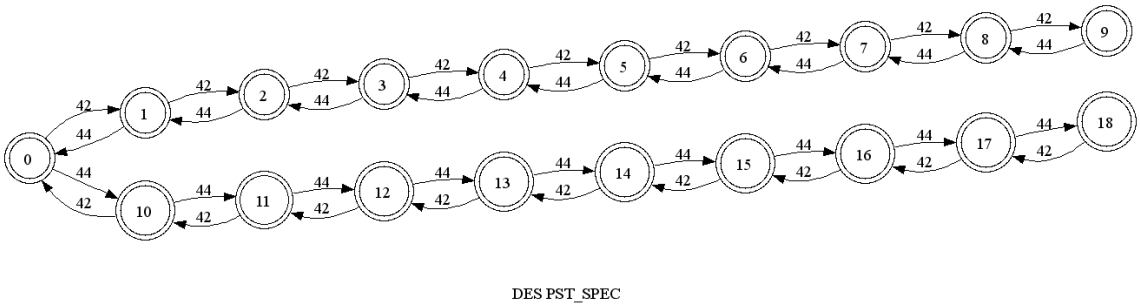


Fig.4.3. Specification for PST E_{PST}

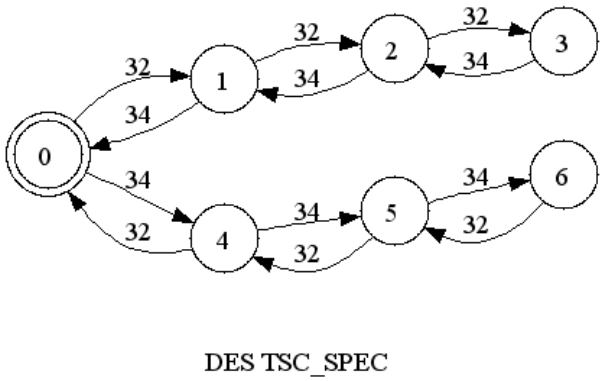


Fig.4.4. Specification for TSC E_{TSC}

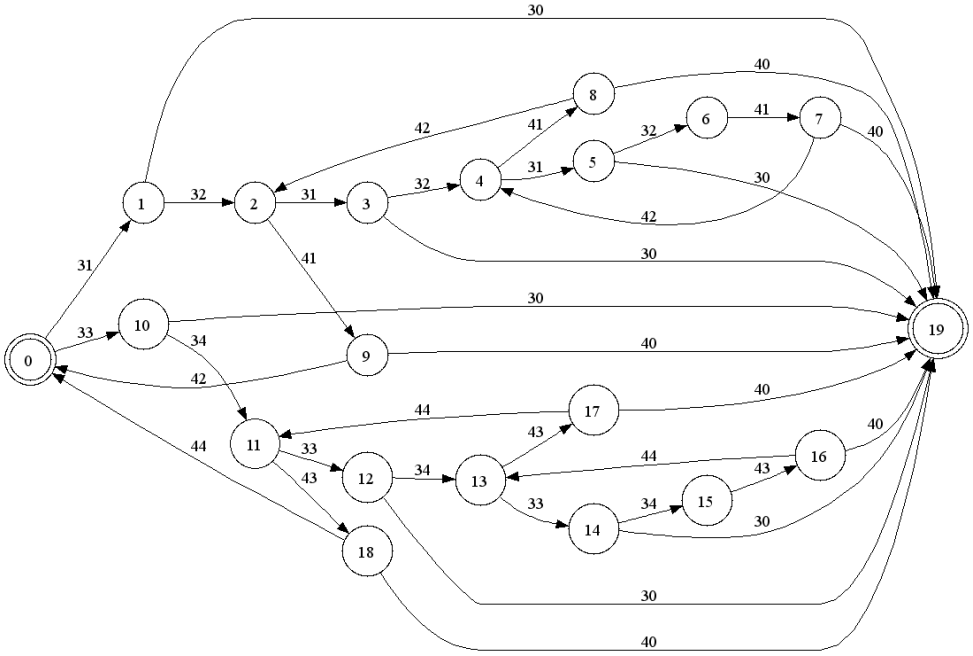


Fig.4.5. Specification for CS2 E_{CS2}

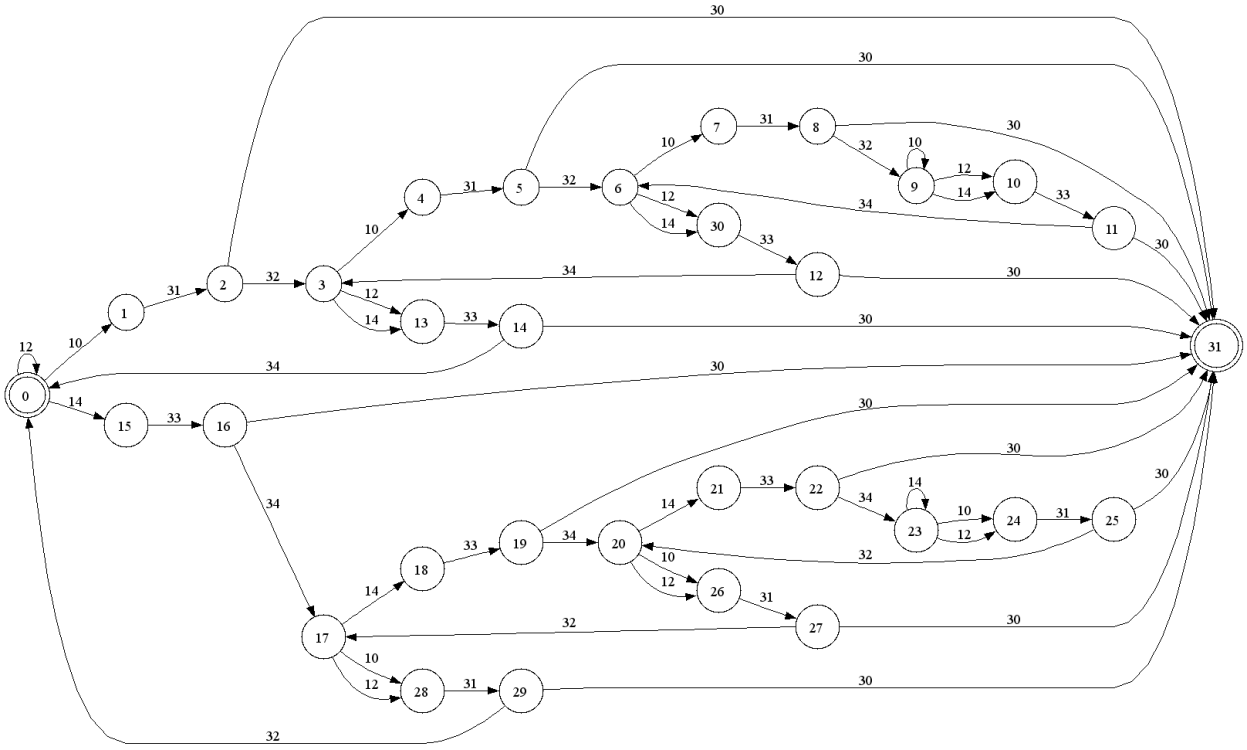


Fig.4.6. Specification for CS1 E_{CS1}

The DES PWRMTR has two states. 0 is a marked state that represents the initial state. 1 represents the state where the meter has to report a increase, decrease or constant supply. TSC_P has three states. 0 is a marked state and represents the condition where no action has to be taken. 1 represents the action of decreasing capacitance. 2 represents the action of increasing capacitance.

The DES PST_P has three states. 0 is a marked state and represents the condition where no action has to be taken. 1 represents the shift-down of taps, thereby decreasing supply. 2 represents the shift-up of taps, thereby increasing supply. The DES TSC_SPEC has 7 states to represent the 7 different combinations of the capacitors. State 0 represents the mean capacitance and is the marked state. States 1, 2 and 3 represent successive decrements of capacitance (and therefore power). States 4, 5 and 6 represent successive increments of capacitance (and therefore power).The DES PST_SPEC has nineteen states which correspond to the 19 different tap positions. State 0 is the mean tap position and is a marked state. States 1-9 represent the lower tap positions (decrease in supply). States 10-18 represent the upper tap positions (increase in supply).The DES PLANT1 has 5 states.

In order to obtain a comprehensive behavioral model that contains information about the plant as well as the desired behavior, we take a parallel composition of the plant and the corresponding specification. By taking a parallel composition, we are taking a union of the event sets of the plants and specification. Hence, we have a comprehensive model. In order to find the plant of a sub-system, the generators of the plants that constitute the sub-system are composed together to form the generators for the local plants/sub-system. The local plants for TSC and PST do not have any other plants in them; hence the generators for PST and TSC remain the same. But, the local plants CS1 and CS2 are the combination of different plants.

Gcs1 = Gpwrmtr || Gtsc

Gcs2 = Gtsc || Gpst

The behavioral models for the local plants are then found as

Elocx = Glocx || Ex

Hence, the behavioral model for TSC is E1=TSC_P||TSC_SPEC. For PST it is PST_P||PST_SPEC. For Control System 1 it is Gcs1||PLANT1. For Control System 2 it is Gcs2||PLANT.

These models specify the required behavior of the plant. Next step is the generation of supervisors working in a feedback architecture which will restrict the plant to the required behavior.

The supervisors for the local plants were generated and their numbers of states were reduced using TCT.

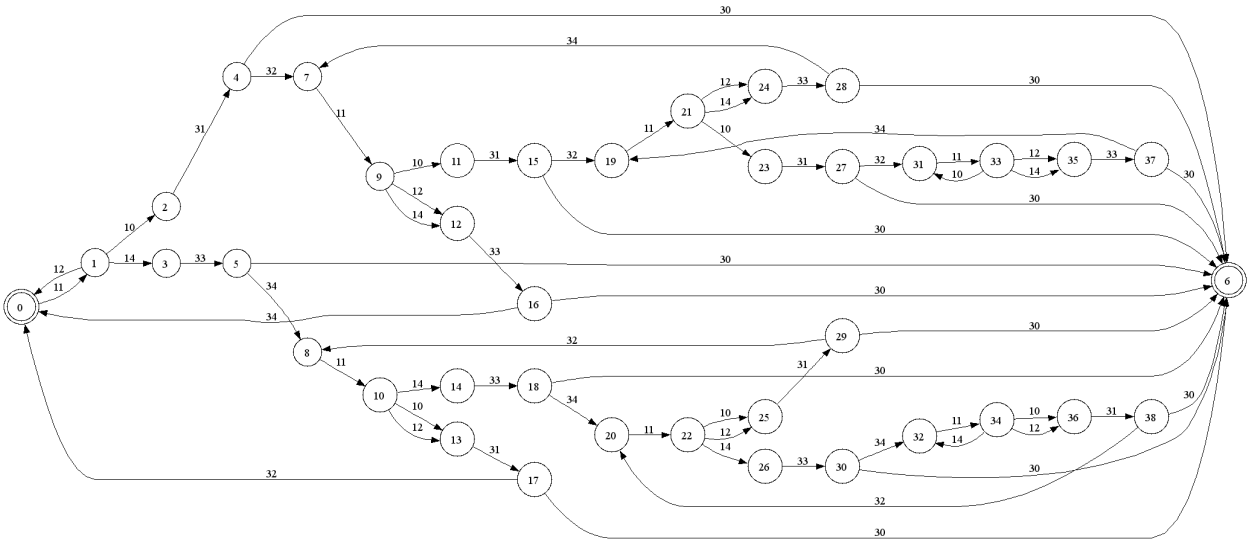


Fig.4.7. Supervisor Control System 1

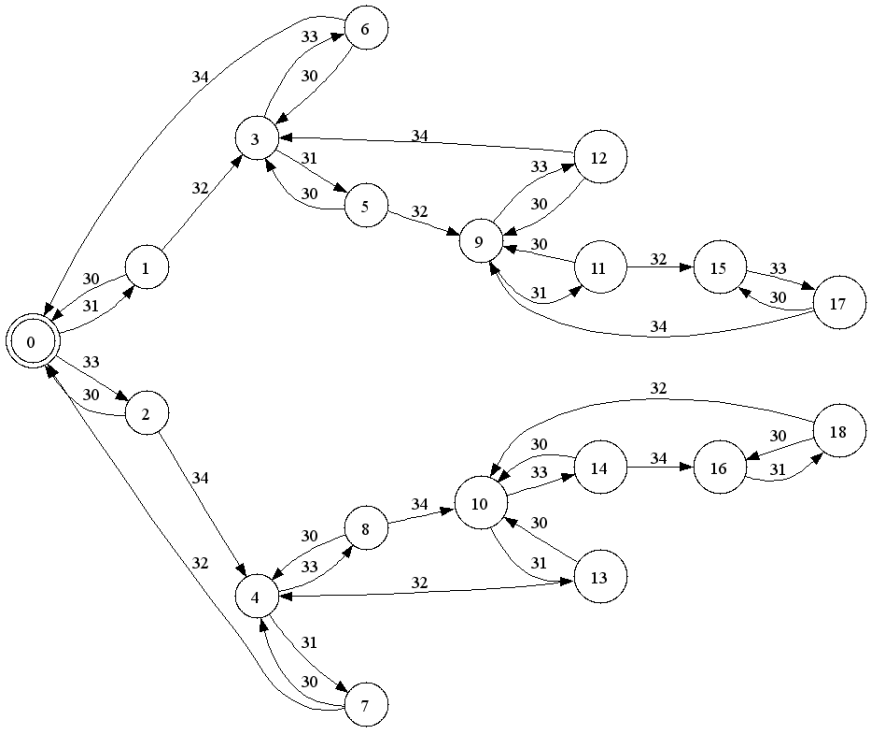


Fig4.8. Supervisor TSC

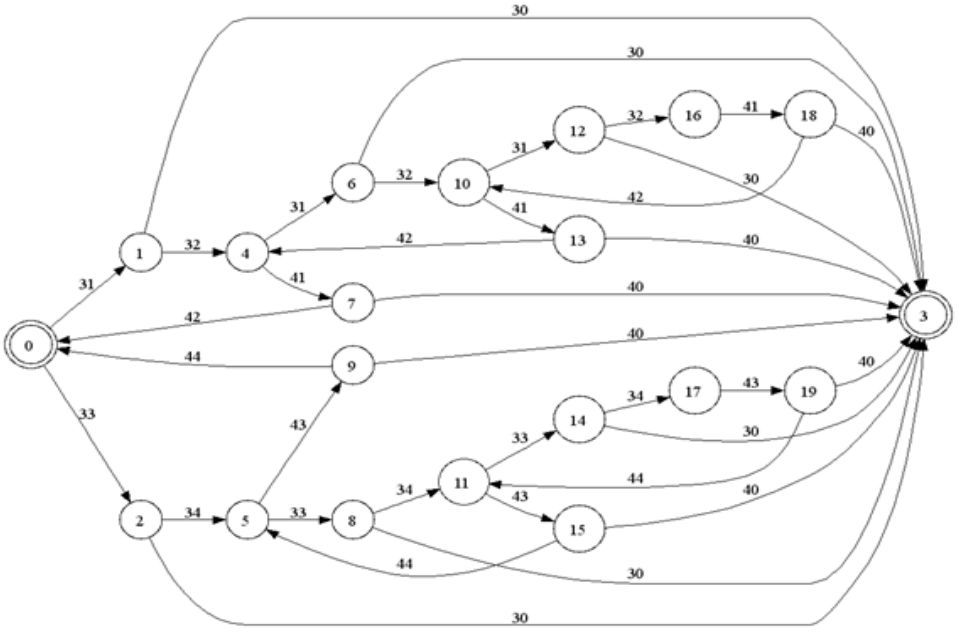


Fig.4.9. Supervisor Control System 2

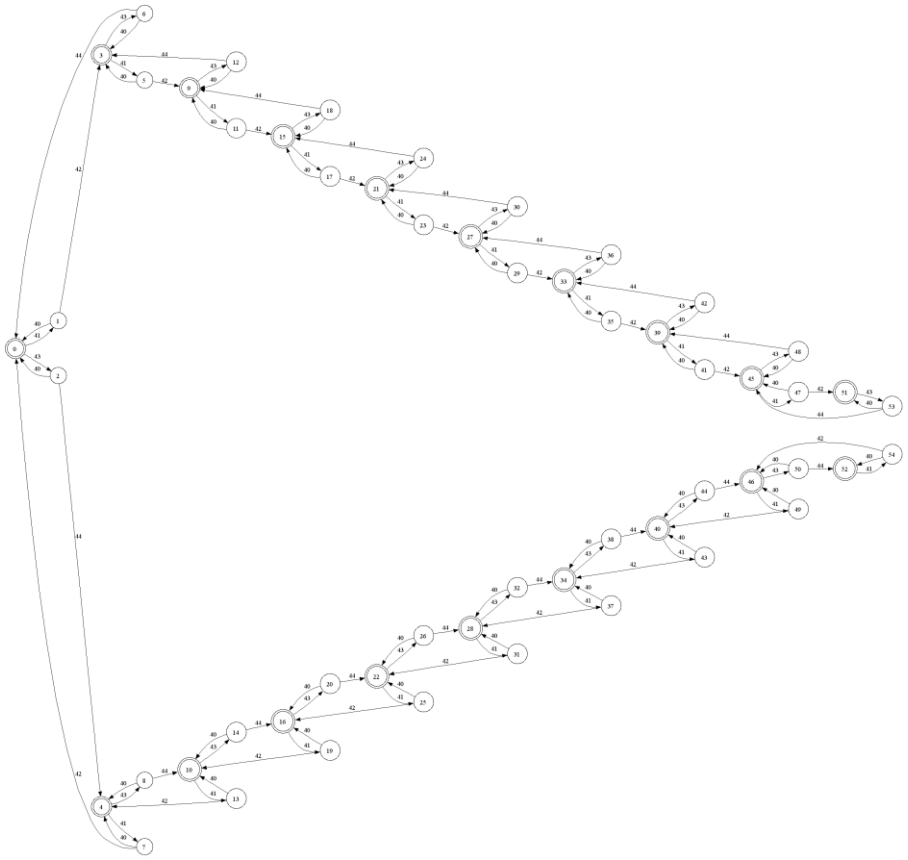


Fig.4.10.Supervisor PST

CHAPTER 5

Automatic Code generation and Simulation

As discussed in Chapter 4, the generators of the plant, their corresponding specifications and their Supervisors have been generatored. We have designed modular supervisors having

CS1 = states 39, transitions 64

TSC = states 19, transitions 36

CS2 = states 20, trnasitions 36

PST = states 55, transitions 108.

Although the total number of states and transitions have drastically reduced in comparison to the monolithic solution (states 4983, transitions 10525), it is still high. The supervisor reduction using TCT also did not help much. But if we look at the supervisor state diagrams for TSC , CS2 and PST we can see that a pattern of the same transitions occur after some states transitions. Hence, these can be reduced to less number of states and a counter can be added to count the occurance of the next similar loop in the supervisors [10].

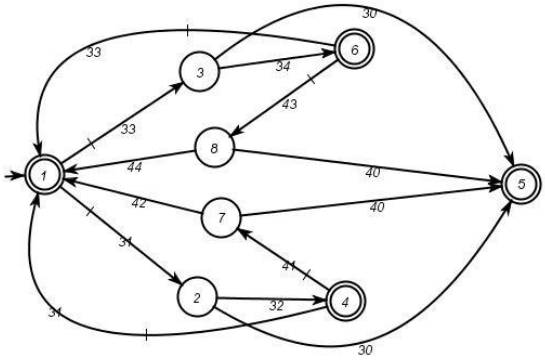


Fig.5.1. Reduced Supervisor for CS2

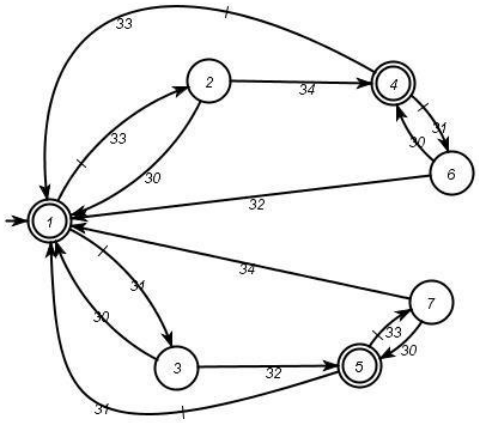


Fig.5.2. Reduced Supervisor for TSC

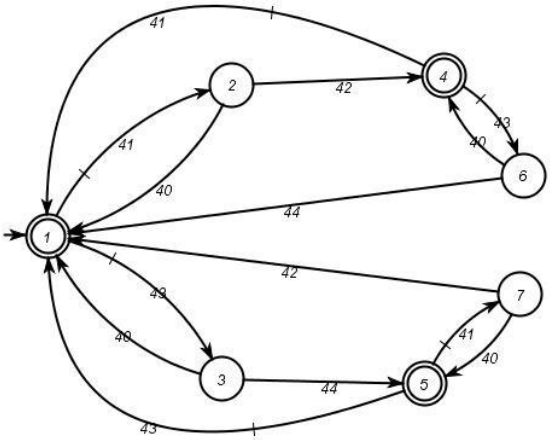


Fig.5.3. Reduced Supervisor for PST

From the above state diagrams it can be seen that if from state 6 event 31 is enabled or from state 4 event 33 is enabled then, the transition to state 0 occurs. A counter is initialized and incremented each time to keep track of these transitions. Similarly, for PST a counter is assigned for transitions from states 4 and 5. The counters are UP-DOWN counters and for TSC and CS2 supervisors they can count ± 3 and for PST they can count ± 9.

After the supervisor reduction, the models of the plants, specifications and the reduced supervisors were imported to IDES3. IDES [12] is a tool with GUI to design DES and perform the major operations on them. We used TCT to model the generators since we found the usage easier and more functionality in TCT. The models were imported to the IDES to as the software ides2st, the automatic code generator we used only detects IDES models. The tool ides2st [2] converts the supervisors and the plant models into PLC structured text code.

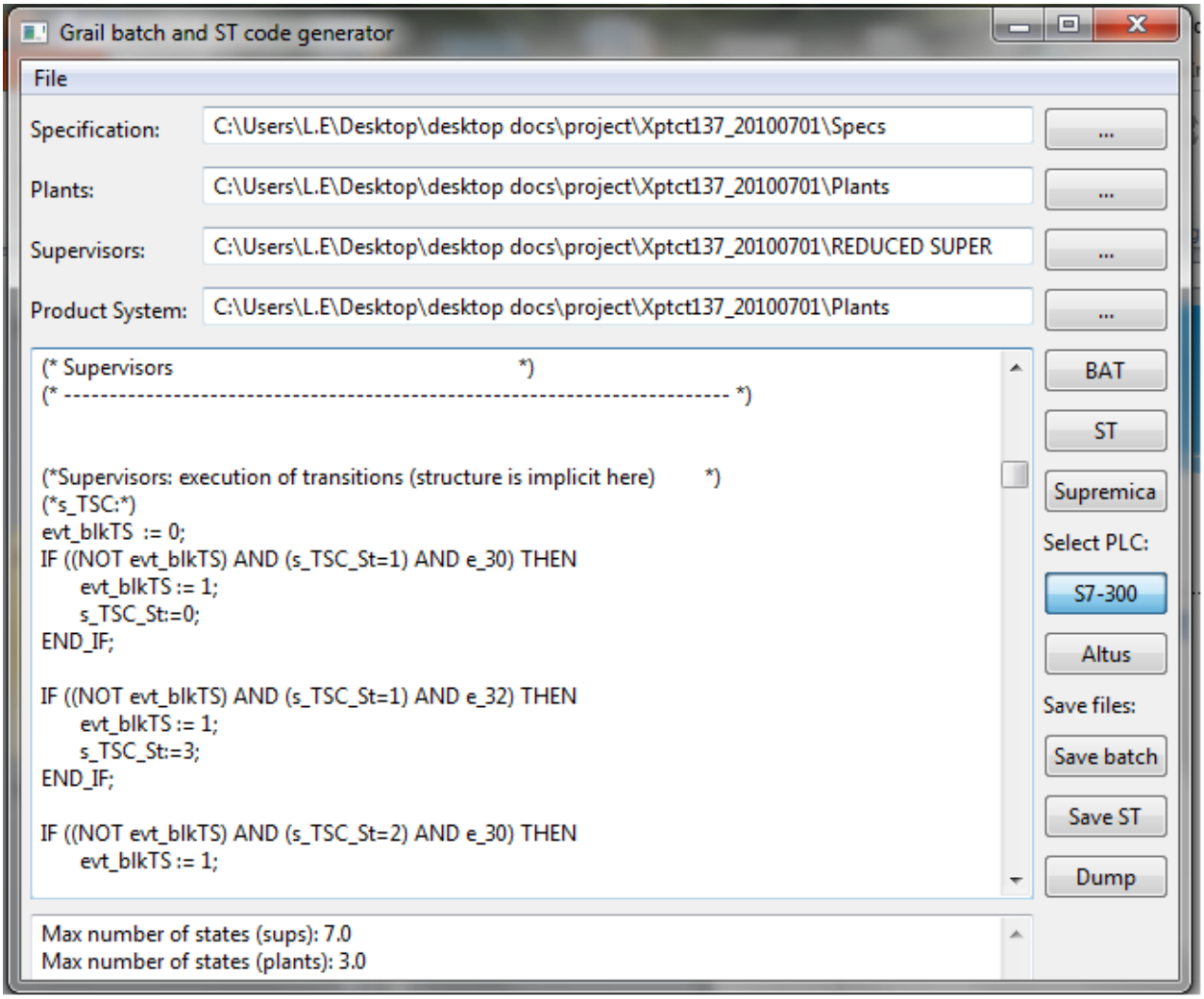


Fig.5.4. ides2st code generation

It assigns a variable to every supervisor state and event. According to ides2St, when an event takes place its event variable is set and remains like that until the supervisor is updated. Ides2st uses an auxiliary variable evt_blk to make sure that only one transition occurs in each cycle. This deals with the problem of avalanche effect and inexact synchronization. It doesn't mean that those subsequent transitions will be ignored by the program, but it will be postponed. Their variables are set, but they just run in a convenient state at the supervisors.

The program is divided into two Function Blocks, one for the supervisors and the other for the product system. The operational procedure level is not generated by ides2st and is left to the programmer. The supervisor level further consists of two subdivisions. First, the FSM of the supervisors are dealt. The second part contains the disabling De_[event] of the controllable events by the supervisor. If De_[event] is set the supervisor will prevent the event from occurring at a certain state. The product system level is also divided into two parts. The first has the uncontrollable event code as uncontrollable events must be updated before the controllable ones. The second contains the controllable events code. Thus before any controllable event can be triggered the code will check if there is any uncontrollable event's bit set. If so, the local plant FSM is updated and sets the response event variable e_[event] to update the supervisor. Were it not, and the local plant is in a state where a controllable event is not disabled, the variable e_[event] bit is set which means a command is sent to start a subroutine and to update the supervisor after this local plant does.

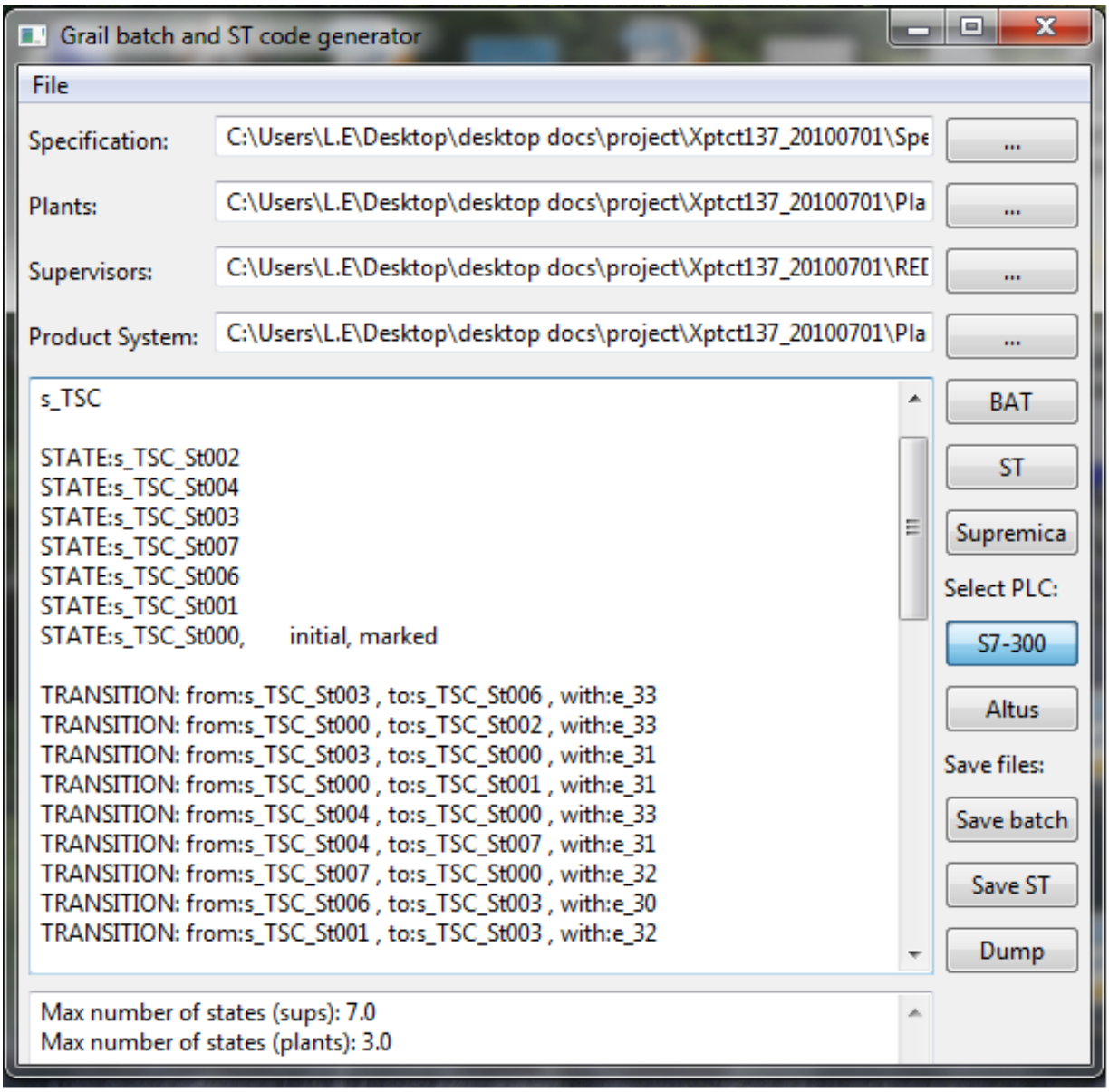
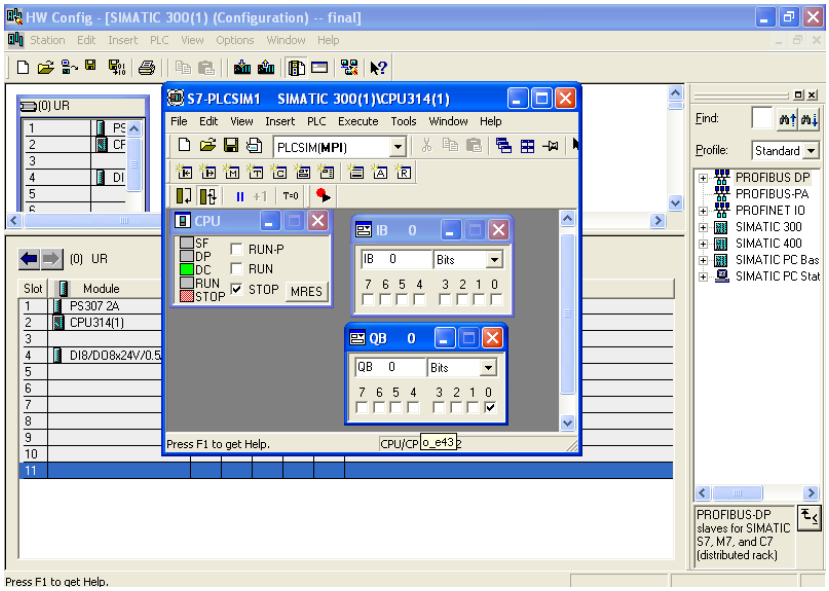


Fig.5.5. ides2st code dumping to check transition list

The above figure shows the **Dump** feature in ides2st. This feature shows all the states and the transitions according to the ST code. From the transition list, the correctness of the code can be checked even before actually applying it on the system.

Simulation

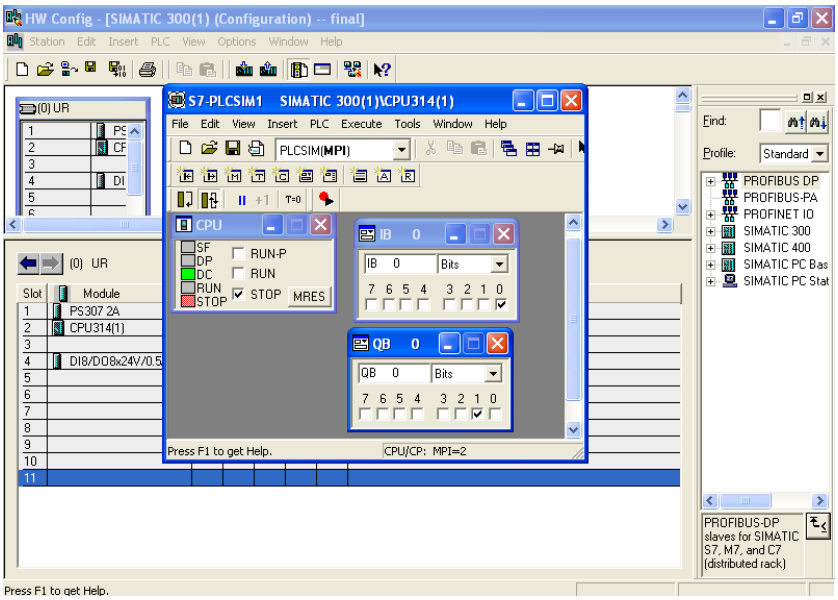
After generating the code using ides2st we added the extra bit of code for the operational procedures, the synchronization of the supervisors and the counters to the ST code. Simatic Step 7 was used to compile the scl source file to generate the statement list file (stl). PLCsim was used to simulate the compiled code.



I0.0 = e_10
I0.1=e_30
I0.2=e_32

Fig.5.6.PLCsim simulation

Q0.0=e_11
Q0.1=e_31
Q0.2=e_33



CHAPTER 6

Results and Discussions

The modeling of a DPFC using discrete event system approach is logical from the fact that the DPFC has a discrete switching nature. Solving the problem using conventional methods would have been confusing and difficult. The modular approach adopted resulted in supervisors with very less number of states making physical implementation possible. Compared to the monolithic solution which had 4983 states [10], the highest number of states a modular supervisor had was 39 which was Control System 1. Generating the supervisors using SCT gave us a formal method to write the PLC code as the PLC code could be directly generated from the supervisor state machines.

Compared to [10], the extra modular supervisor we had introduced reduces the code length as we can see that the CS2 have repetitive states for which a counter can be used. The automatic code generation using ides2st simplified the PLC code part further. It saved us a lot of time and effort. Also, by automatic code generation we could be sure that there were no typing errors. Also by checking the models and remodeling time and again we obtained an optimum solution and could be sure that there were no errors in the modeling part. We have also addressed some problems regarding the PLC implementation of supervisors. The code generator provides solutions to some of the problems like avalanche effect and inexact synchronization. The rising edge of signals was detected to avoid taking the signal into consideration in two consecutive scans even if the signal hadn't changed.

Finally we conclude that the supervisors were optimum, non-conflicting and could control the plant efficiently. This approach can be used for any plant following the discrete event nature. Hence our work can be taken as an initiative to prove that industrial application of SCT is possible. Software for complete automation of modeling and code generation could be developed in the future to simplify the process further.

REFERENCES

[1] Queiroz M.H. and Cury J.E.R. Synthesis and Implementation of Local Modular Supervisory Control for a Manufacturing Cell,in *Proceedings of the 6th International Workshop on Discrete Event Systems*, Zaragoza, Spain,(2002): pp. 377-382.

[2] Queiroz M.H and Silva Y.G. Formal Synthesis, simulation and automatic code generation of a Factory Manufacturing Cell,in *20th International Congress of Mechanical Engineering*, Gramado, Brazil,(2009).

[3] Ramadge P.J.G and Wonham W.M. The Control Of Discrete Event Systems, *Proceedings of IEEE*,77-1(1989): pp. 81-98.

[4] Vieira A.D and Cury J.E.R and Qeiroz M.H. A Model for PLC Implementation of Supervisory Control of Discrete Event Systems , *IEEE*(2006).

[5] Fabian M. and Hellgran A. PLC-based Implementation of Supervisory Control for Discrete Event Systems,in *Proceedings of the 37th IEEE Conference on Decision & Control*, Tampa, Florida, USA,(1998).

[6] Malik P., Generating Controllers from Discrete-Event Models, in *Proceedings of Summer School in Modelling and Verification Processes (MOVEP)*, Cassez F. et al., Eds., 2002, pp. 337-342.

[7] Dietrich P., Malik R., Wonham W.M., and and Brandin B., Implementation Considerations in Supervisory Control, in *Synthesis and Control of Discrete Event Systems*.: Kluwer

Academic Publishers, 2002, pp. 185-201.

[8] Balemi S. "Control of Discrete Event Systems: Theory and Application," Swiss Federal Institute of Technology, Switzerland, PhD Thesis 1992.

[9] Cassandras C.G. and Lafortune S., *Introduction to Discrete Event Systems* New York, USA: Springer, 2008.

[10] Afzalian A. and Noorbakhsh M. and Nabavi S.A. PLC Implementation of Decentralized Supervisory ,in *17th IEEE International Conference on Control Applications,* San Antonio, Texas, USA,(2008).

[11] Feng L. and Wonham W.M. TCT: A computational Tool for Supervisory Control Synthesis,in *Proceedings of 8th International Workshop on Discrete Event Systems*, Ann Harbor, USA,(2006): pp. 388-389.

[12] Rudie K. The Integrated Discrete-Event System Tool,in *Proceedings of 8th International Workshop on Discrete-Event Systems*, Ann Harbor, USA,(2006): pp. 394-395.

[13] Afzalian A., Nabavi Niaki S.A., and Irani R. and Wonham W.M. Discrete-Event Systems Supervisory Control for a Dynamic Flow Controller, *IEEE Transactions on Power Delivery*,24-1(2009).

[14] Brandin B.A. The Real-time Supervisory Control of an Experimental Manufacturing Cell , *IEEE Transactions on Robitics and Automation*,12-1(1996): pp. 1-14.